# System-on-Chip implementation of a PV dynamical Reconfiguration Algorithm

S. Ciaglia, Eric Monmasson, G. Petrone, G. Spagnuolo

# System-on-Chip implementation of a PV dynamical Reconfiguration Algorithm

S.Ciaglia[1], E.Monmasson[2], G. Petrone[1], G. Spagnuolo[1]
[1]University of Salerno, Dep. of Informatics, Electrical Eng. and Applied Mathematics - Italy
[2]Université de Cergy-Pontoise - SATIE - France
E-mail: saraciaglia@gmail.com, emonmasson@u-cergy.fr, gpetrone@unisa.it, gspagnuolo@unisa.it

***Abstract*—Dynamical reconfiguration of is receiving attention from the scientific community because of the advantages it can ensure with respect to module-dedicated electronics when partial shadowing of the photovoltaic field occurs. Indeed, lower cost and conversion losses as well as improved diagnostic capabilities make this solution of high interest for industrialization in a near future. Some stochastic algorithms have been proposed recently for determining the best electrical connection among the panels that ensures the highest power production for the actual shadowing pattern. In this paper, any novel reconfiguration algorithm is proposed, but the optimized implementation of recently proposed approach is described. The advantages achieved by using a system-on-chip platform, through a suitable exploitation of the features it offers, are highlighted. The experimental results confirm that the use of a system-on-chip platform gives significant benefits for the proposed application, because the parallelization of the algorithm allows to achieve great execution time reduction with respect to what can be achieved through its standard software implementation.**

## I. Introduction

System-on-chip (SoC) architectures address many processing needs, by integrating microprocessors and reconfigurable logic into a single chip, by offering a wide bandwidth that is not achievable by each one of the two chip solutions and low power compared to an equivalent solution built on the two chips (software modules and high-performance hardware). This technology has gained benefits from the advantages of FPGA such as flexibility, small circuit size and IP reusability. Application examples can be found in differentiated areas such as data processing [1], artificial vision [2], ambient intelligence [3], motion planning [4], indoor localization [5], learning algorithms [6], speech recognition [7], power converter [8], ac drives [9].

A further application that can benefit from SoC features is the dynamical reconfiguration of photovoltaic (PV) arrays. This means that the electrical connection among the panels is not fixed once for all, but the PV panels terminals arrive to a switching box, which is equipped with a processor running a suitable control algorithm, so that the connection thereof is changed a number of time per day, depending on the actual operating conditions of the PV panels. The advantages offered by this technique are widely described in [10]. Very recently a preliminary version of a deterministic approach, which is aimed at determining the electrical connection among the panels for a given shadowing pattern, has been presented [11]. Instead, more consolidated methods in literature rely on stochastic methods; for instance, in [12] the best configuration of the PV panels in a two paralleled strings PV array is searched for. In [13], the same approach was improved and a preliminary validation was obtained by using a classical digital platform. In [13] the effort was done in improving the algorithm ability of exploring the space of the solutions, without a special attention to the reduction of the computation time. Indeed, no special care was given to the calculation of the objective function, which requires the evaluation of the Maximum Power Point (MPP) power delivered by each candidate electrical configuration of the PV panels in the field. Indeed, the algorithm [13] takes as input the sampled current vs. voltage (I-V) curves of all the panels of the PV field: it is assumed that all the panels, one at one time, are disconnected for some milliseconds for acquiring the I-V curve, thus with a small impact on the power production. It is also reasonably assumed that the load is constant or, at least, an energy buffer decouples the PV field from the load during such operation. Afterwards, for each possible electrical configuration, the evolutionary algorithm has to compute the I-V curve of the candidate solution, which corresponds to a possible electrical connection of the available PV panels. For a real application, e.g. consisting of 24 PV panels distributed into 2 parallel connected strings, if each PV panel I-V curve has been digitally acquired and consists in 100 points, it is evident that a high number of complex operations has to be performed.

The objective of this paper is not the introduction of a novel reconfiguration algorithm, but to address the problem of reducing the computation time of the algorithm presented in [13]. Thus it is firstly assumed that the I-V curves of all the PV panels in the field, which are supposed to be 24, have been experimentally acquired with a given number of samples. This assumption is reasonable, because the reconfiguration system needs a periodical information about the working conditions of the panels in order to take a decision concerning the need of reconfiguring the electrical connection or not. On the basis of the curves samples, the evolutionary algorithm computes the power vs. voltage (P-V) curve and its MPP of each candidate configuration, so that the best one is determined and the PV panels electrical interconnections are settled according to it. Because of the high computational effort required by the P-V curve computation for each candidate solution, the adoption of a SoC platform is helpful: the processor takes

care of the algorithm core and the FPGA fabric is used for running in parallel the evaluation of the P-V curve of a number of candidate solutions. Indeed, each P-V curve evaluation is independent from the others, because it corresponds to a unique combination of the 24 panels in the two available strings. The number of instances to run in parallel would be dependent on the available resources.

In the following sections a description of the used SoC platform is given. The main implementation features are described in section III. Then, implementation results are presented and discussed in section IV. Finally, conclusions are drawn in section V.

## II. Zynq-7000 All Programmable SoC Platform

The target platform is a Zedboard [14] (Zynq Evaluation and Development), development board based on the Xilinx Zynq-7000 all programmable SoC [15]. This class of SoCs integrates a processing system (PS) based on a dual-core ARM Cortex-A9 and a Xilinx programmable logic (PL) based on a the Artix-7 logic embedded into a single device.

The Zynq processing system has four functional blocks, comprising the Application Processor Unit (APU), I/O peripherals, memory interfaces, and a multilayered ARM AMBA AXI interconnect. APU includes dual ARM Cortex-A9 MPCore CPUs with ARMv7 and includes level-1 and level-2 caches. This dual processor approach supports various software implementation allowing the use of a single operating system running in SMP (Symmetric Multiprocessing) mode, the use a couple of operating system in AMP (Asymmetric Multiprocessing) mode and the use of one operating system in BMP (Bound Multiprocessing). Each core can work with a frequency maximum of 667 MHz with a speed of 2.5 DMIPS/MHz (Millions Instructions per Second with Dhrystone benchmark). Moreover, each core has a FPU (Floating Point Unit) engine with a computing capacity of 2.0 MFLOPS/MHz (Million Floating Point Operations Per Second) and a NEON media processing engine.

The size of resources of the the Z7020 are summarised in Table I.

Table I: PL resources

| Logic slices | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|
| 13300 | 53200 | 106400 | 140 | 220 |

The Zynq board provides communication between programmable logic and processing system through the standard ARM AMBA AXI bus, which is capable of handling Master-Slave multiple connections and offers high bandwidth coupling between the two parts. The programmable logic is connected to the processing system through multiple AXI interfaces: two 32-bit AXI master (GP master) ports, two 32-bit AXI slave (GP slave) ports, and four 64-bit high-performance AXI slave (HP) ports. Figure 1 shows a block diagram of the Zynq-7000 architecture.

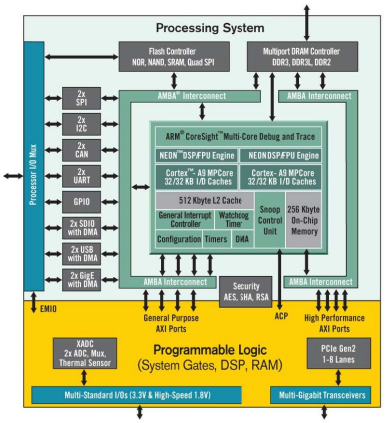



Figure 1: Zynq 7000 architecture

Zynq is based on a centric processor architecture, thus the processing system does not require the configuration of the programmable logic and the PL does not need to be powered on. The board supports booting from NOR, NAND, Quad-SPI, SD, or JTAG. A single processor core copies the first stage boot loader (FSBL) from primary boot device, selected through the jumpers that are positioned at the top of the board, to the internal on chip memory and executes the code. The configuration bit stream which contains optional programmable logic programming data is downloaded from a memory location into the PL.

## III. Evolutionary Algorithm Implementation

The evolutionary algorithm considered in this paper is exactly that one described in detail in [13]. In brief, the gene represents one of the 24 panels in the PV field and it is an integer number assuming the values $\{0, 1, 2\}$, stating that the panel is not connected to any strings, thus it is excluded from the resulting PV field and in $0$ status, or it is in the first string and in $1$ status, or it is in the second string and in the $2$ status. The individual is then an array of 24 integer numbers, each one describing the status of one panel in the PV field. Thus, the individual is a tentative solution of the problem: it describes the electrical configuration of the 24 panels in two parallel connected strings, or disconnected. As said above, it is assumed that the I-V curves of each panel have been experimentally acquired and consist of a given number of samples. Given an individual and the panels' I-V curves, the I-V curve of the PV field described by that individual is calculated by simply applying Kirchhoff current and voltage laws: PV panels in series have the same current and strings in parallel share the voltage. Thus, three computation loops are needed: one for determining the voltages of the first string for any current value of each panel belonging to it. Another for the same operation concerning the second string. The third computation loop is requested for calculating the whole PV field current values for each voltage value of the two strings in parallel. This short description explains why the computation of the objective function, which is the MPP, for each individual

is so heavy if even few tens of samples describe the I-V curve of each PV panel. The evolutionary algorithm characteristic functions, thus the selection, crossover, mutation operators, have a computational load that is incommensurably lower than the one of the objective function. Thus, the latter needs some specific approach if the algorithm running time has to be shortened in order to make the reconfiguration approach more appealing for practical applications. Indeed, after the PV panels I-V curves have been acquired, during the time needed for computing the new best electrical configuration to be settled the old configuration runs, this meaning that some energy is missed. The more the new best configuration is different from the old one, the higher the amount of power that is not produced during the calculation time needed by the processor.

In Zynq architecture the programmable logic allows to extend the capabilities of the processing system by hosting additional hardware accelerators, which can take advantage of the hardware parallelism offered by the FPGA. Thus, this type of SoC platform gives the greatest advantages when PS and PL cooperate. A preliminary analysis has been conducted in order to achieve an efficient sharing of the PV reconfiguration process through software and hardware resources. The computational task having the greatest impact on the performance metric and that is more suitable for the implementation on PL has been evaluated as first. The execution time has been chosen as the metric to measure the performance gain. This is motivated by the fact that the execution time is relatively easy to measure and also because response time is the most important factor for the application, since the decrease of the system response time allows to identify the best connection among the PV panels in the shortest possible time, thus increasing the power produced by the PV field. In order to analyze potential benefit of hardware accelerators the full PV reconfiguration algorithm has been firstly implemented in software in one of the ARM processor (C code). Thus, the resulting execution time is serving as reference for the wholes forthcoming Hw/Sw implementations.

The evolutionary algorithm, which is a Genetic Algorithm (GA), has two main parameters whose values greatly affect the algorithm performances: the population size and number of digital samples describing the experimentally acquired I-V curves of the panels. Population size is one of the parameters affecting the convergence of the algorithm towards the optimal solution. The number of samples acquired in the panel I-V curve affects the capability of detecting all the MPPs, appearing when the panel is subjected to mismatching effects, with a suitable accuracy. The higher the number of samples in the panel curves, the higher the computation time needed for calculating the I-V curve of a candidate array configuration. In [16] a method for downsampling the panels I-V curve without affecting the accuracy in the MPPs detection was introduced. This was used in [13] and it is adopted for achieving the results shown in this paper as well. In the following sections, case A will indicate when the panel I-V curve is described by 100 points and case B when the panel I-V curve includes 25 points. Table II summarizes, for both cases A and B, the fully software implementation performance by showing the required amount of clock cycles and the percentage of time spent by each operation of the stochastic algorithm with regard to the total execution time.

Table II: Execution time by using the software approach

| | Number of clock cycles | | % | |
|---|---|---|---|---|
| **Operation** | **case A** | **case B** | **case A** | **case B** |
| Selection | 52681 | 52314 | 0.02 | 0.118 |
| Crossover | 16735 | 16662 | 0.006 | 0.038 |
| Mutation | 33506 | 32975 | 0.012 | 0.074 |
| Fitness eval | 260077208 | 44364006 | 99.96 | 99.77 |
| **Total** | 260180130 | 44465957 | 100 | 100 |

Table II shows that case B execution time is significantly lower than case A, but the table also shows that in both cases the execution time of the fitness evaluation is greater than 99% of the total execution time of the algorithm. Operations as selection, crossover and mutation affect only the 0.2% of the overall processing. This confirms that the function deserving to be implemented within Hw accelerators is the fitness function, which includes the computation of the I-V curve of the candidate configuration. Figure 2 shows the architecture of the proposed SoC-based parallel algorithm. The main part of the algorithm (called Master), which is implemented in the PS, stores and manages the population by making selection, crossover and mutation operations. As for the fitness evaluation modules (called Slaves), they are implemented in the PL.

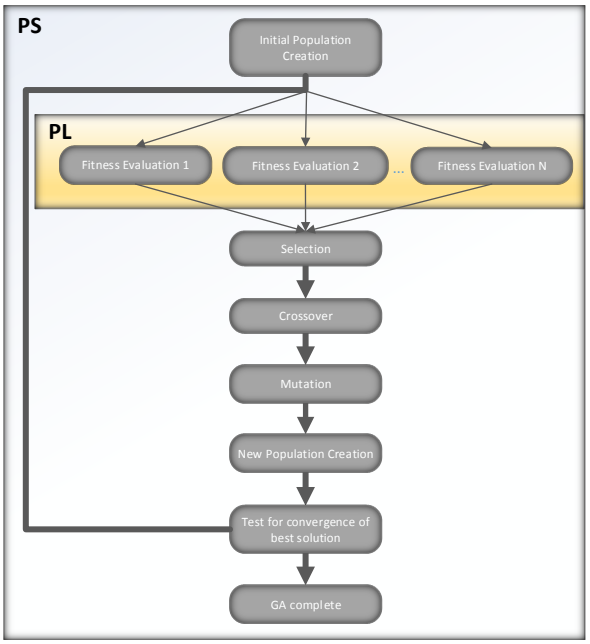



Figure 2: Master / Slave architecture for the proposed SoC-based parallel GA

C code has been used to generate the fitness evaluation IP module, by means of the high level synthesis tool Vivado HLS. Its main functionality is to accelerate the process of transforming an algorithm written in C, C++ or System C into an IP core inside FPGA. The fitness evaluation IP module is verified as an independent entity in Vivado HLS by using test benches written in C language. The first accelerator has been created by a direct synthesis of the Vivado HLS tool based on the original C code used in the fully software version of the algorithm. This basic accelerator does not provide an improvement if compared with the the fully software version implemented in the ARM processor: the maximum number of fitness evaluation blocks in parallel can be only two and communications between the two IP modules and the processor are very slow. This is shown in Figure 3.

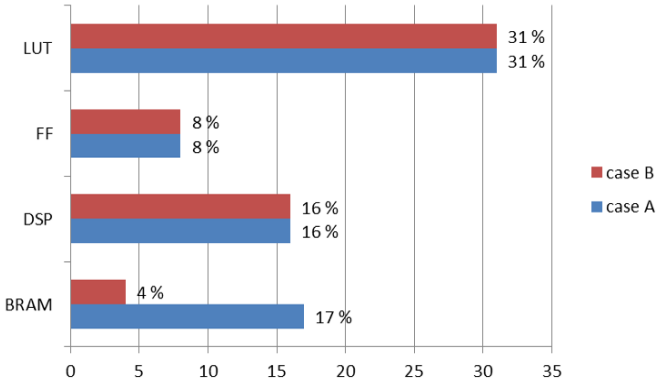



Figure 3: Resource utilization in the case of floating point accelerators

A key limitation of this basic accelerator is in the cost of implementing the floating point arithmetic operations required by the evaluation of the candidate solutions I-V curves. An efficient representation of real values is required and a fixed point representation allows to decrease hardware area and execution time. An analysis has thus been performed to choose the optimal number of bits to represent the fractional part of the variables of the algorithm in order to minimize the consumed resources while maintaining a sufficient level of accuracy. The first factor that has been considered is the data format error, which is calculated as the average distance between the fixed and the floating point representation of the fitness value in percent. The consumed resource and the precision is calculated for each fixed-point representation. It can be seen from Figure 4 that a data format higher than 12 bits allows keeping the precision of the calculation below 1% which is enough for this application.

The consumed resources and the percentage of clock cycles needed to complete the fitness evaluation execution of one individual, by varying fixed point precision have been also evaluated. Figure 5 shows that the resources used with fixed point representation with the fractional part in the range comprised between 12 and 16 bits are almost the same. Figure 6 shows that there is a difference of 400 thousands clock cycles

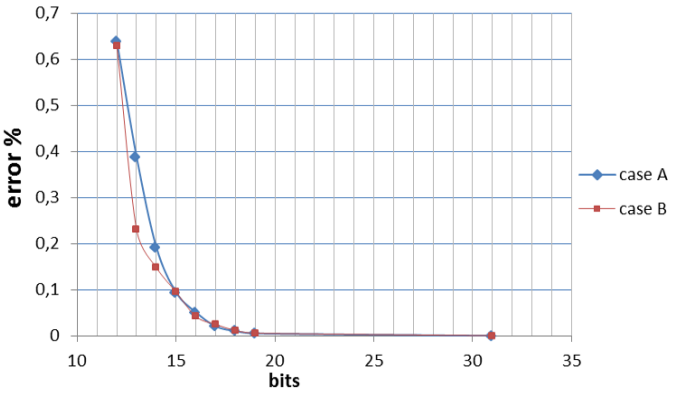



Figure 4: Error increase by varying fixed point precision

in case A and 100 thousand in case B when the number of bits is increased from 12 to 13. Thus, it appears that a 12 bits representation is the best compromise in terms of accuracy, rapidity and consumed resources.

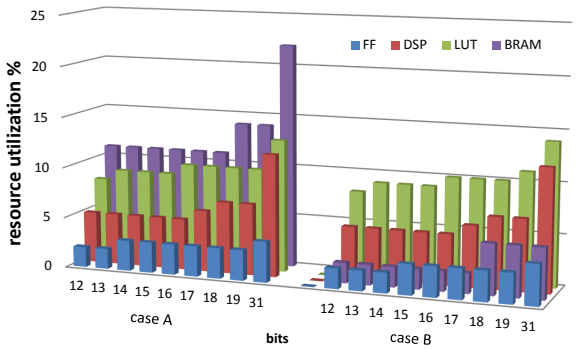



Figure 5: Resource utilization increase vs fixed point precision

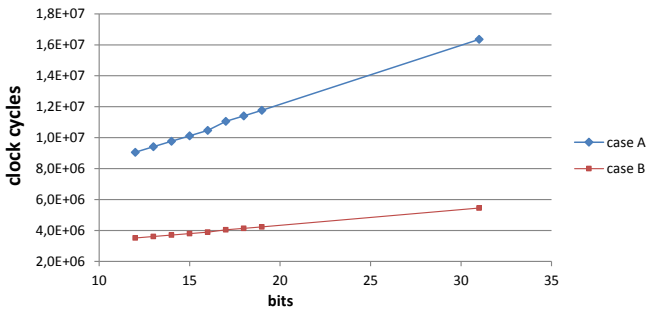



Figure 6: Clock cycles increase vs fixed point precision

Figure 7 shows that the transition from a floating point representation to a 12 bits fixed point representation has significantly decreased the consumed resources. For the two considered examples, the number of FFs, LUTs and DSPs is reduced by more than 70%. Moreover, the execution time needed to complete the fitness evaluation of one individual is reduced by more than 40% with respect to the case of the floating point representation (see Table III).

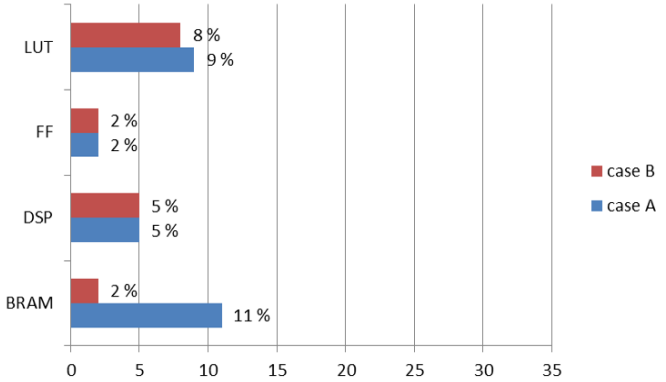


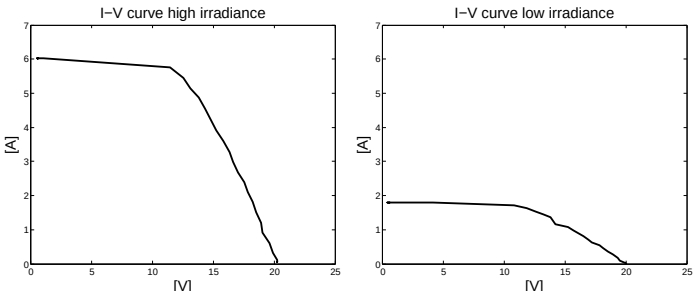

Figure 7: Resource utilization for a 12 bits fixed point accelerators

Table III: Number of clock cycles needed to execute the fitness IP module

| | **Case A** | **Case B** |
|---|---|---|
| **Floating point** | 16581278 | 6116079 |
| **Fixed point** | 9058506 | 3514506 |

## IV. Implementation results

A case study including 24 panels to be reconfigured in two strings connected in parallel is considered. It is assumed that 12 panels have the experimentally acquired I-V curves shown in Figure 8 (a) and other 12 panels have the ones depicted in Figure 8 (b). It is worth to note that the approach proposed in this paper has the aim of accelerating the process of determining the best configuration of the PV panels regardless of the particular shadowing condition affecting the array, thus of the shape of the I-V curves representing the actual operating conditions of the PV panels. The time needed for the calculation of a single objective function value depends on the number of samples of the I-V curves and not on the operating conditions of the PV panels. Indeed, the objective function associated to a GA individual gives the value of the maximum power delivered by the PV field configuration described by that individual. Because of the fact that the PV field is supposed to be organized in two parallel connected strings of PV panels, the objective function calculation requires, as first step, the computation of the I-V curves of the both strings. For each one the algorithm scans the I-V samples of the panels, on a current basis, thus summing the panels voltages for each current value. Afterwords, the resulting I-V samples of the two strings are scanned, on a voltage basis, for calculating the total current as a sum of the currents delivered by each string for that voltage value. Finally, the voltage and current vectors referring to the whole PV field I-V curve have to be multiplied, one by one, in order to obtain the maximum PV field power value. Because of the fact that the panel and string I-V curves were not be sampled with the same current and voltage values, the interpolation between adjacent samples is necessary. This explains the high computation time requested for the evaluation of the objective function of each GA genotype.




(a) *high irradiance* (b) *low irradiance*

Figure 8: Experimental I-V curves of the PV panels of the array

In the proposed SoC controller architecture the software part performs the control of the whole system with several specific tasks, namely, GA operations, signal conditioning, system monitoring, while each accelerator in the hardware part performs fitness evaluation of each individual. Some implementations, differing in the number of parallel fitness evaluation IP modules, have been run.

Vivado IDE is used to create the SoC architecture, to perform the whole hardware development flow, to generate the bitstream and the report files containing the resources consumed by varying the number of the fitness evaluation blocks that are implemented in parallel. Vivado supports easy IP integration into the Zynq by offering the graphical environment IP Integrator. This tool is able to connect blocks of Vivado IP catalog with end user IP modules, provided that they meet the IP-XACT standard. The fitness evaluation block, created with Vivado HLS, is imported into Vivado IP catalog and it is integrated into the Vivado project since Vivado HLS packages IP, using IP-XACT standard. Xilinx SDK tool is used to generate the software that implements the master operations running on the ARM cortex A9 processor in order to evaluate the acceleration factor, which is calculated as the ratio between the execution time of a full software implementation and the execution time an architecture which includes hardware accelerators implemented via the HLS design tool. The execution time is measured by accessing to the global timer processor register.

The tests were conducted for both cases A and B. For each case and for all different hardware implementations, the used GA parameters, shown in Table IV were applied.

Table V shows the percentage of consumed resources in terms of DSP, slice and BRAM for case A. It puts in evidence that the maximum number of blocks that can be put in parallel is 3 since a higher number of blocks would saturate the available number of BRAMs. In this case the amount of data to be stored is significant and each fitness evaluation block requires 30% of the total BRAM resources.

Table IV: GA parameters

| Parameter | Value |
|---|---|
| Number of panels | 24 |
| Population size | 48 |
| Maximum generation number | 100 |
| Stall generation number | 25 |
| Crossover probability | 0.7 |
| Mutation probability | 0.2 |

Table V: Resource Utilization case A

| # of accelerators | DSP % | Slice % | BRAM % |
|---|---|---|---|
| 1 | 10.45 | 17.45 | 30 |
| 2 | 20.91 | 32.87 | 58.57 |
| 3 | 31.36 | 47.78 | 87.85 |

As for the execution time and the accelerating factor of the SoC implementation, Figure 9 shows that the accelerating factor follows a linear trend. By reaching the maximum allowable number of hardware fitness evaluation modules, an acceleration factor of 7.1 times compared to the performance of the basic processor implementation is obtained. The total execution time is 64 s.

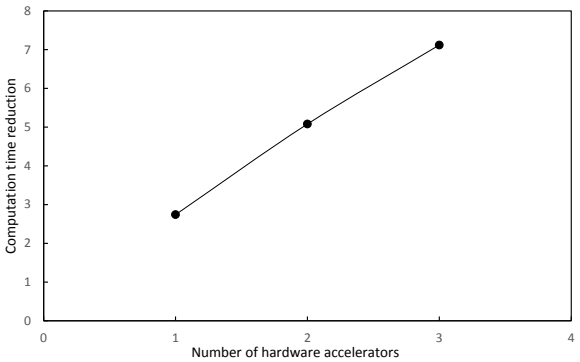



Figure 9: Accelerating factor case A

Table VI shows the percentage of consumed resources for case B, where it is obvious that the reduced resource consumption of each evaluation block allows implementing more hardware accelerators in parallel compared to case A, up to a number of 8.

Table VI: Resource Utilization case B

| Number of accelerators | DSP % | Slice % | BRAM % |
|---|---|---|---|
| 1 | 3.63 | 13.35 | 7.14 |
| 2 | 7.27 | 25.53 | 14.29 |
| 4 | 14.54 | 45.73 | 28.57 |
| 6 | 21.82 | 69.54 | 42.86 |
| 8 | 29.09 | 87.59 | 57.14 |

Figure 10 shows that the accelerating factor is comparable to case A, but the execution times are significantly smaller. The system speed increases by a factor of approximately 10 by reducing the number of input samples by a factor of 4.

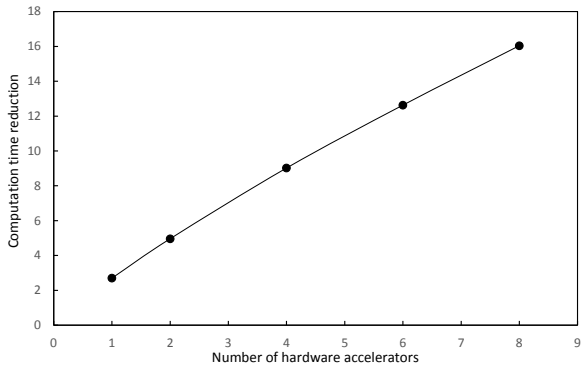



Figure 10: Accelerating factor case B

Figure 10 also shows that the overall accelerating factor follows a linear trend but the greater the number of hardware accelerators the worse the efficiency of the master slave architecture. Indeed, if by using only one hardware accelerator the computation time reduces by a factor of 2.7, the expected computation time with eight parallel blocks, with a linear trend, would be 21.6. Instead, it is 16, so that the linear trend is missed because of the additional communication overhead.

## V. Conclusions

In this paper a system-on-chip implementation of an evolutionary algorithm for photovoltaic dynamical reconfiguration is presented. The hardware implementation of the objective function has allowed to accelerate the execution time significantly compared to a full software implementation, which is beneficial for the extraction of the maximum power from the photovoltaic array. Implementation results obtained on a Zynq show that the objective has been fully achieved leading to a significant reduction of the execution time. The execution time of the whole GA algorithm for case A and B is, respectively, 65.8 s and 2.7 s with the maximum parallelization achieved, while the one obtained using a full software implementation is 468.6 s in case A and 43.4 s in case B. Further work is in progress for achieving a further level of parallelization also inside the routine which computes the single objective function. This operation would lead to a further reduction of the computation time and exploitation of the hardware resources.

## References

[1] V. Sklyarov and I. Skliarova, "High-performance implementation of regular and easily scalable sorting networks on an fpga," *Microprocessors and Microsystems*, vol. 38, pp. 470 – 484, 2014.

[2] H. Tabkhi, M. Sabbagh, and G. Schirner, "Power-efficient real-time solution for adaptive vision algorithms," *IET Computers & Digital Techniques*, vol. 9, pp. 16 – 26, 2014.

[3] I. del Campo, K. Basterretxea, J. Echanobe, G. Bosque, and F. Doctor, "A system-on-chip development of a neurofuzzy embedded agent for ambient-intelligence environments," *IEEE Trans. Syst. Man Cybern. Part B: Cybern.*, vol. 42, no. 2, pp. 501 – 512, 2012.

[4] C.-C. Tsa, H.-C. Huang, and S.-C. Lin, "Fpga-based parallel dna algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 1016 – 1026, 2011.

[5] J. Rodrguez-Arajo, J. J. Rodrguez-Andina, J. Faria, and M.-Y. Chow, "Field-programmable system-on-chip for localization of ugvs in an indoor ispace," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1033 – 1043, 2014.

[6] I. del Campo, J. Echanobe, G. Bosque, and J. M. Tarela, "Efficient hardware/software implementation of an adaptive neuro-fuzzy system," *IEEE Trans Fuzzy Syst.*, vol. 16, no. 3, pp. 761 – 778, 2008.

[7] S.-T. Pan and X.-Y. Li, "An fpga-based embedded robust speech recognition system designed by combining empirical mode decomposition and a genetic algorithm," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 9, pp. 2560 – 2572, 2012.

[8] E. Ormaetxea, J. Andreu, I. Kortabarria, U. Bidarte, I. M. de Alegria, E. Ibarra, and E. Olaguenaga, "Matrix converter protection and computational capabilities based on a system on chip design with an fpga," *IEEE Trans. Power Elettr.*, vol. 26, no. 1, pp. 272 – 287, 2011.

[9] I. Bahri, L. Idkhajine, E. Monmasson, and M. E. A. Benkhelifa, "Hardware/software codesign guidelines for system on chip fpga-based sensorless ac drive applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2165 – 2176, 2013.

[10] G. Spagnuolo, G. Petrone, B. Lehman, C. Ramos Paja, Y. Zhao, and M. Orozco Gutierrez, "Control of photovoltaic arrays: Dynamical reconfiguration for fighting mismatched conditions and meeting load requests," in *IEEE Industrial Electronics Magazine*, March 2015, pp. 62–76.

[11] M. Orozco Gutierrez, J. Ramirez Scarpetta, C. Ramos Paja, G. Petrone, and G. Spagnuolo, "Optimized configuration of mismatched photovoltaic arrays," *IEEE Journal of Photovoltaics*, in press.

[12] P. Carotenuto, S. Curcio, P. Manganiello, G. Petrone, G. Spagnuolo, and M. Vitelli, "Algorithms and devices for the dynamical reconfiguration of pv arrays," in *Int. Exhib. Conf. Power Electron., Intell. Motion, Renewable Energy Energy Manage., Nuremberg, Germany*, May 2013.

[13] P. Carotenuto, A. D. Cioppa, A. Marcelli, and G. Spagnuolo, "An evolutionary approach to the dynamical reconfiguration of photovoltaic fields," *Neurocomputing*, vol. 170, pp. 393 – 405, 2015.

[14] *Zedboard Hardware User's Guide*, 2nd ed., Avnet, 2014.

[15] *Zynq-7000 All Programmable SoC Technical Reference Manual (UG585)*, 1st ed., Xilinx, 2015.

[16] P. Carotenuto, P. Manganiello, G. Petrone, and G. Spagnuolo, "Online recording of a pv fingerprint," *IEEE J. Photovoltaics*, vol. 4, no. 2, pp. 659 – 668, 2014.